Candle®

Triggering MQSeries Channels in Distributed Systems

A Candle MQSeries Survival Guide

Triggering MQSeries Channels in Distributed Systems

By Wayne Bucek, Candle Corporation

One of the most significant challenges at data processing installations today is cross- platform inter-networking. Driven by the availability of cheap MIPS on non-mainframe platforms, information technology (IT) executives are aggressively pursuing the use of distributed platforms for new application development. This has led to an increased number and variety of platforms that must be supported.

Prior to messaging and queuing technology, sophisticated conversational communication programs were needed to enable cross-platform connectivity. With the introduction of MQSeries,, IBM has delivered a more viable solution to the cross-platform, inter-networking complexities that many organizations face today.

Channels play a vital role in the distributed queuing component of MQSeries. This article compares triggering channels to manually started channels, and examines the implementation specifics of triggering channels.

MESSAGE CHANNELS

Distributed Queuing (DQM) is the component of MQSeries that enables messages to be sent to remote systems. Message channels are the communication "links" used by the distributed queuing component. These links are logical connections between the nodes of the MQSeries network. The logical connections are built upon the existing network infrastructure. MQSeries channels are compatible with TCP/IP, SNA, DECnet and Netbios communication protocols. The capability to function in all of the prevalent networking protocols in use today is a major point of difference between MQSeries and other messaging solutions.

Channels are unidirectional and, accordingly, are defined in pairs. This pairing of channels allows messages to flow in either direction between two MQSeries queue-managers. Channels come in four flavors: sender, receiver, server, and requester. These four channel types can be combined to form one of five types of channel pairs. Generally speaking, the definable channel pairs operate as follows.

- In a *Sender - Receiver* channel, the sender channel in one system starts up and sends messages from the transmission queue to the receiver channel in the remote system.
- In a *Requester - Server* channel, the requester channel in one system starts up to receive messages from the remote system, and then starts up the server channel in the remote system.
- In a *Requester - Sender* channel pair, the requester is used to start up the sender channel. The sender channel immediately terminates the call, then restarts the channel according to the specifications in its channel definition file. This channel pair is referred to as a callback channel.
- In a *Server - Requester* channel pair, the server channel has a defined communication link. The server channel can be opened by the requester, or can initiate communication with the requester by itself.
- In a *Server - Receiver* channel configuration, the server must contain the communication link definition, and channel startup must be initiated at the server end of the link.

While each combination has distinct advantages, the Sender - Receiver pair is by far the most popular.

STARTING CHANNELS

Channels must be started before any messages can flow. MQSeries provides three methods of issuing the commands which start channels: entering control commands from the command line, entering MQSC commands from within the RUNMQSC utility, and using programmable command formats (PCF) on supported platforms.

Channels can be started immediately by issuing the RUNMQCHL command from the command line. This command causes a local instance of the Message Channel Agent (MCA) to begin execution. The MCA receives the name of the channel to be started as parameter input from the RUNMQCHL command. The MCA then reads the channel definition file directly to obtain its attributes and starts the channel. The appropriate instance of an MCA must be executing on each side of the communications link for the MQSeries channel to start successfully.

RUNMQSC is a utility that allows users to interactively issue MQSeries commands. Once the RUNMQSC command has been issued, operational commands can be sent to control and display MQSeries resources. The START CHANNEL command causes the designated channel to start in the same manner as RUNMQCHL.

PCFs define command and reply messages that can be exchanged between a program and any queue manager in a network. PCF messages are sent to the SYSTEM.ADMIN.COMMAND.QUEUE. The MQSeries command server processes these messages and sends reply messages to queues defined by the Reply-to-Q and Reply-to-Qmgr fields in the message descriptor. PCF commands are used to implement systems management functions, including the starting and stopping of channels and channel initiators.

WHY TRIGGER CHANNELS

Channels can be started dynamically, based on the presence of a message in the associated transmit queue. This method, which is referred to as triggering channels, requires the use of some advanced MQSeries facilities. Triggering channels offers benefits not available with manually starting channels.

Reduced operational requirements

The majority of users select triggering channels for reduced operational requirements. As discussed earlier, channels must be started for messages to flow. If channels are started manually (via RUNMQCHL), it is reasonable to assume that this is done one time, immediately after queue manager initialization. However, channels rely upon external resources to function properly. Error conditions like unavailable remote queue managers, underlying network problems, or operating system failures will prevent an MQSeries channel from starting, or subsequently cause it to fail.

The necessary response to problems of this nature varies greatly, depending on which mechanism has been chosen to start the channel. For manually started channels, operator intervention is required to reissue the START CHANNEL or RUNMQCHL command. This must be done after the underlying environmental or networking issues have been resolved.

Coordinating these efforts might not be as easily accomplished as first thought. In a distributed queuing environment, this would likely involve operations personnel at geographically disparate locations. To make matters worse, in today's climate of networked businesses, it would not be unusual to find that operations personnel from different companies need to be involved to resolve the problem.

Consider the use of triggered channels. Triggered channels require the use of a channel initiator process. A channel initiator is a special purpose trigger monitor that starts the channel MCAs based on the presence of a message in the associated transmit queue. This process runs independently of MQSeries resources. As long as messages exist on the transmit queue, the channel initiator will try to start the channel. If the MQSeries objects and external resources required for channel operation are available, the channel initiator can successfully start the channel. When the external resources needed for a channel to start are unavailable, the channel initiator's attempt to start the channel fails.

Further Complications

Version 5 of MQSeries has introduced many significant performance enhancements in MCA operation. These features have been widely discussed, and as such, will not be covered here. However, along with these enhancements come some behavioral changes for channels. MQSeries version 5 MCAs retain channel status across invocations. Certain network errors are classified as permanent errors by the MCA.

When a permanent error occurs, the receiver channel retains a status of STOPPED. In these cases, normal triggering alone cannot successfully restart the channel. A START CHANNEL command must be explicitly issued on the receiver side of the channel to clear the "hard fail" condition that was recorded.

On the sender side of the channel, you are likely to encounter a channel status of RETRYING. This condition indicates that the MCA is attempting to restart the channel, but does not have the required resource available to it. It is also likely that the XMITQ being serviced by the sender side MCA was placed in a "get inhibited" state when the original channel failure occurred. An 'ALTER QLOCAL' command must be issued to MQSeries to reset the GET attribute of the XMITQ to ENABLED.

Disconnect Interval

One aspect of triggering channels is the channel disconnect interval. This interval dictates how long a channel will stay in the RUNNING state while idle. When this timer is exceeded, MQSeries sets the channel status to INACTIVE. When there are new messages to be sent, the channel initiator restarts the channel.

Care must be taken to properly specify the disconnect interval. If the disconnect interval is set too short, CPU resources will be wasted in needlessly shutting down and then quickly restarting channels. Conversely, setting this parameter value too long defeats its useful purpose, in most cases.

Shedding Some Windows Lite on the Subject

MQSeries version 2.1 for Windows95/NT, also known as the Windows Lite queue manager, adds some interesting twists to channel operation. For starters, triggering is not supported by this release of MQSeries. Accordingly, triggered channels are not supported on the platform. Be aware that the channel disconnect interval defaults to a value of 6000 seconds. After 100 minutes of inactivity, the channel will be placed in the INACTIVE state. However, since the Lite queue manager does not support triggering, manual intervention must be taken to restart the channel. Rather than promoting unnecessary manual intervention, set the value of DISCINT to zero. This disables disconnect processing, thereby enabling the channels to remain in the RUNNING state for the duration of the active MQSeries for Windows connection.

As a side note, remember that the Lite queue manager does not support a dead letter queue. Undeliverable messages will cause the channel to shut down every time.

Regardless of how you start channels, the simple fact remains that they must get started. In a large MQSeries network, automating the startup of channels becomes essential. Without using triggering channels, a significant effort in custom automation is required. Further consider that MQSeries exists across many different platforms. Different automation products would be required to implement the automation needed to start the channels. Coordinating this automation would also be a substantial undertaking. Contrast this with triggering channels. Channel initiators exist on all level two MQSeries platforms. Definitions common to all MQSeries platforms are used to control their execution. Finally, there is no custom coding required to implement triggering channels.

THE DETAILS

Following are detailed definitions required to implement triggering channels. See Figure 1 for the triggered channel flow.

Channel definition

The channel definition contains the attributes that describe a channel. This information is contained in the channel definition file (CDF). The CDF must be local to the system where the MCA is starting. The MCA reads the CDF at startup time to implement a given instance of a channel. Following is an example of a channel definition:

```
DEFINE CHANNEL(QM1.TO.QM2) CHLTYPE(SDR) +
DESCR('Sender channel to QM2') +
CONNAME(QM2) TRPTYPE(TCP) XMITQ(QM2) CONVERT(YES)
```

Transmission queue definition

Each MQSeries channel is associated with a transmission queue. Transmission queues contain messages that are destined for remote queue managers. The channel reads messages out of the transmission queue and transmits them to the MCA on the remote side of the link. The remote MCA then delivers the message to the destination queue (assuming the queue exists on the remote platform).

A transmission queue is defined as a local queue with the usage attribute set to *transmission*. For a triggering channel, the TRIGGER parameter must be specified. This specification causes MQSeries to place a trigger record in the corresponding initiation queue. This is the record that the channel initiator reads as input when starting a channel.

Finally, in releases of MQSeries prior to version 5, a PROCESS definition must be associated with the transmission queue. In normal usage (application processing versus channel triggering), the process definition identifies the path to a program that will be executed when the trigger condition is met. In the context of channel initiation, the process definition is not used to identify a program for execution, but to specify the name of the channel to be started.

In MQSeries version 5 systems, the name of channel to be started should be provided in the trigdata field on the transmission queue definition. This replaces the requirement for a dummy process definition.

Local queue definitions

Use the following to define the transmit queue to version 2 queue manager QM2:

```
DEFINE QLOCAL(QM2) USAGE(XMITQ) + TRIGGER TRIGTYPE(FIRST) INITQ(IQ) PROCESS(P1)
```

Use the following to define the transmit queue to version 5 queue manager QM2:

```
DEFINE QLOCAL(QM2) USAGE(XMITQ) + TRIGGER TRIGTYPE(FIRST) TRIGDATA('QM1.TO.QM2') INITQ(IQ)
```

Define the initiation queue associated with QM2:

```
DEFINE QLOCAL(IQ) DESCR ("Initiation queue for QM2")
```

And define the process definition (P1) to be used as input by the channel initiator:

```
DEFINE PROCESS(P1) USERDATA(QM1.TO.QM2)
```

Starting the channel initiator

The final piece of the puzzle is starting the channel initiator process. This is accomplished with the run channel initiator command, RUNMQCHI, that specifies the name of the initiation queue. An example of starting the channel initiator for the default queue manager is:

```
RUNMQCHI -q IQ
```

Alternatively, the MQSC facility can be used to issue the START CHINIT, again specifying the name of the initiation queue. The MQSeries System Management Guide for the appropriate platform provides details of the ways to run the channel initiator, along with other control commands.

CONCLUSION

The distributed queuing component of MQSeries provides organizations with a vehicle to enable cross platform application connectivity. MQSeries channels are at the heart of DQM. When an organization has many channels defined, the operational complexity of managing them can be a daunting issue. Fortunately for users, MQSeries provides comprehensive facilities such as triggering to manage channels automatically and efficiently across platforms.

BIBLIOGRAPHY

IBM MQSeries System Administration, sc33-1873-00
IBM MQSeries Distributed Queuing Guide, sc33-1139-07.
IBM MQSeries for Windows NT Systems Management Guide, sc33-1643-00.
Messaging & Queuing Using the MQI, Burnie Blakely, Harry Harris and Rhys Lewis, McGraw-Hill, Inc., 1995.

Some Valuable MQSeries Tips

Here are a few invaluable tricks of the trade prepared by Candle's Consulting & Services team…

UNIX

When using MQSeries on a Solaris platform, you must increase the amount of shared memory from the normal default. However, while increasing shared memory allocation improves the MQSeries performance, too large an increase robs the kernel of physical memory and can slow the entire machine.

When using an MQSeries queue manager on the same machine with a database server, both will compete for shared memory resources.

MQSeries V5 uses the POSIX threading library. When building threaded applications it is important to ensure that all products use the same thread library for their threading model.

MQ applications must be built on Sun Solaris using the C compiler located in /opt/SUNWspro/bin/cc; the version located in /usr/ucb/cc will build applications unable to connect to the Queue Manager.

NT & Windows

When installing MQSeries on a machine which is part of an NT Domain, ensure that the MQ administrator id is also part of the "mqm" group on the Domain Controller; otherwise you will be unable to create a Queue Manager.

Do not delete the "mqm" user, even if it does not conform to corporate naming standards; MQSeries will cease to function correctly.

For MQSeries Clients running under Windows 95, the MQSERVER environment variable will take precedence over the MQCHLTAB and MQCHLLIB environment variables.

On NT, it is possible to specify a BAT file in the APPLICID of the PROCESS definition, which can be useful in setting environment variables before running an application, or can be used to run multiple applications from a single trigger.

It is not documented that before you uninstall MQseries on NT, you must go into Control Panel / Services and stop the MQSeries Service manually.

AS/400

When installing MQSeries for AS/400, v3r7 on AS/400 machine with OS v4r1 PTF #SF46028 is required to enable communication with other platforms.

Roman-8 code should not be used as the default code set when communicating an AS/400 and a HP/UX. Instead, set ISO8859-1 (CCSID 819) as the default code.

Tandem NSK

To bypass limits of some home terminal devices use the Virtual Hometerm Services (VHS) to simulate a physical home terminal device - this product has a very high limit of the number of supported home terminals.

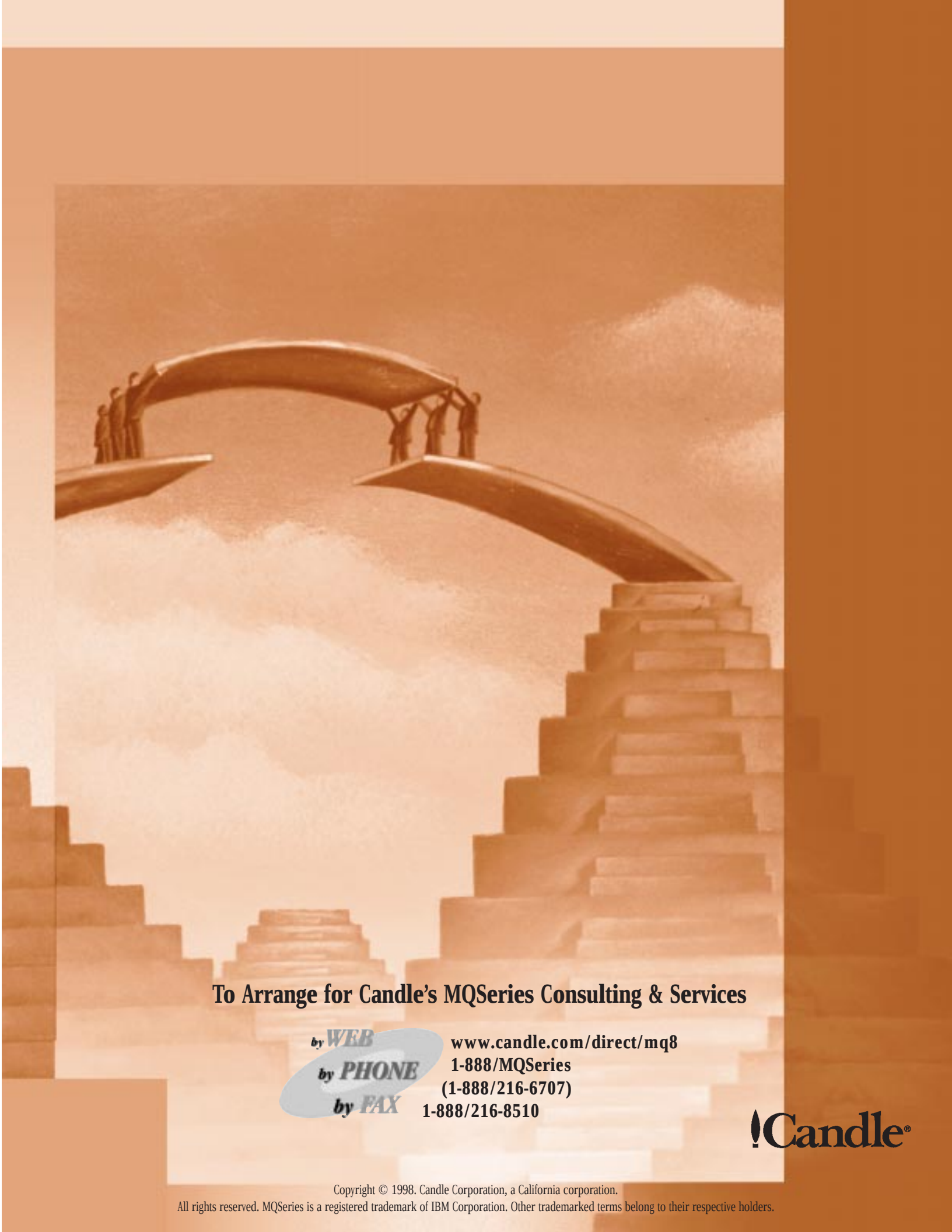
The AUTOSTART channel attribute (not provided for other platforms) should be used to configure SNA channels to allow remote initiation.

MVS

The IBM supplied Windows 95/NT/OS2 Client Trigger Monitor (RUNMQTRMC.EXE) will not work when triggering from MVS Queues, as the mainframe version of MQSeries will only allow APPLTYPE in the PROCESS definition to be set to CICS, IMS or MVS, and the Client Trigger Monitor needs it to be set to WINDOWSNT, OS2, etc. The only solution is to custom build a Trigger Monitor.

General

When designing an MQSeries-based system that uses Tuxedo to coordinate transactions between multiple XA resource managers, each of your services should invoke only one of the resource managers. For example, if you need to include both Oracle and MQSeries operations in the same transaction, design two services, one for the Oracle operations and one for the MQSeries operations. One of the services should invoke the other using the tpcall mechanism.



To Arrange for Candle's MQSeries Consulting & Services

by WEB www.candle.com/direct/mq8
by PHONE 1-888/MQSeries (1-888/216-6707)
by FAX 1-888/216-8510

!Candle®